October 1998

Parametric Approach to the Specification and Analysis of Real-time System Designs based on ACSR-VP

Hee-Hwan Kwak
University of Pennsylvania

Insup Lee
University of Pennsylvania, lee@cis.upenn.edu

Oleg Sokolsky
University of Pennsylvania, sokolsky@cis.upenn.edu

Postprint version. Published in *Electronic Notes in Theoretical Computer Science*, Volume 25, 1999, pages 38-49, Proceedings of the 1998 ARO/ONR/NSF/DARPA Monterey Workshop on Engineering Automation for Computer Based Systems.
Publisher URL: http://dx.doi.org/10.1016/S1571-0661(04)00130-6

Parametric Approach to the Specification and Analysis of Real-time System Designs based on ACSR-VP

Abstract
To engineer reliable real-time systems, it is desirable to discover timing anomalies early in the development process. However, there is little work addressing the problem of accurately predicting timing properties of real-time systems before implementations are developed. This paper describes an approach to the specification and analysis of scheduling problems of real-time systems. The method is based on ACSR-VP, which is an extension of ACSR, a real-time process algebra, with value-passing capabilities. Combined with the existing features of ACSR for representing time, synchronization and resource requirements, ACSR-VP can be used to describe an instance of a scheduling problem as a process that has parameters of the problem as free variables. The specification is analyzed by means of a symbolic algorithm. The outcome of the analysis is a set of equations and a solution to which yields the values of the parameters that make the system schedulable. These equations can be solved using integer programming or constraint logic programming. The paper presents the theory of ACSR-VP briefly and an example of the period assignment problem for rate-monotonic scheduling. We also explain our current tool implementation effort and plan for incorporating it into the existing toolset, PARAGON.

Keywords
real-time scheduling, formal method, parametrized analysis, process algebra

Comments
Postprint version. Published in *Electronic Notes in Theoretical Computer Science*, Volume 25, 1999, pages 38-49, Proceedings of the 1998 ARO/ONR/NSF/DARPA Monterey Workshop on Engineering Automation for Computer Based Systems.
Publisher URL: http://dx.doi.org/10.1016/S1571-0661(04)00130-6

Parametric Approach to the Specification and Analysis of Real-time System Designs based on ACSR-VP *

Hee-Hwan Kwak, Insup Lee, and Oleg Sokolsky
Department of Computer and Information Science
University of Pennsylvania
Philadelphia, PA
lee@cis.upenn.edu, {heekwak,sokolsky}@saul.cis.upenn.edu

July 19, 1999

Abstract

To engineer reliable real-time systems, it is desirable to discover timing anomalies early in the development process. However, there is little work addressing the problem of accurately predicting timing properties of real-time systems before implementations are developed. This paper describes an approach to the specification and analysis of scheduling problems of real-time systems. The method is based on ACSR-VP, which is an extension of ACSR, a real-time process algebra, with value-passing capabilities. Combined with the existing features of ACSR for representing time, synchronization and resource requirements, ACSR-VP can be used to describe an instance of a scheduling problem as a process that has parameters of the problem as free variables. The specification is analyzed by means of a symbolic algorithm. The outcome of the analysis is a set of equations and a solution to which yields the values of the parameters that make the system schedulable. These equations can be solved using integer programming or constraint logic programming. The paper presents the theory of ACSR-VP briefly and an example of the period assignment problem for rate-monotonic scheduling. We also explain our current tool implementation effort and plan for incorporating it into the existing toolset, PARAGON.

1 Introduction

The desire to automate or incorporate intelligent controllers into control systems has lead to rapid growth in the demand for real-time software systems. Moreover, these systems are becoming increasingly complex and require careful design analysis to ensure reliability before implementation. Recently, there has been much work on formal methods for the specification and analysis of real-time systems [8, 12]. Most of the work assumes that various real-time systems attributes, such as execution time, release time, priorities, etc., are fixed *a priori* and the goal is to determine whether a system with all these known attributes would meet required safety properties. One example of safety property is schedulability analysis; that is, to determine whether or not a given set of real-time tasks under a particular scheduling discipline can meet all of its timing constraints.

The pioneering work by Liu and Layland [17] derives schedulability conditions for rate-monotonic scheduling and earliest-deadline-first scheduling. Since then, much work on schedulability analysis has been done which includes various extensions of these results [11, 28, 25, 4, 26, 22, 18, 3]. Each of these extensions expands the applicability of schedulability analysis to real-time task models with different assumptions. In particular, there has been much advance in scheduling theory to address uncertain nature of timing attributes at the design phase of a real-time system. This problem is complicated because it is not sufficient to consider the worst case timing values for schedulability analysis. For example, scheduling anomalies can occur even when there is only one processor and jobs have variable execution times and are nonpreemptable. Also for preemptable jobs with one processor, scheduling anomalies can occur when jobs have arbitrary release times and share resources. These scheduling anomalies make the problem of validating a priority-driven system difficult. Clearly, exhaustive simulation or testing is not practical in general except for small systems of practical interest. There have been many different heuristics developed to solve some of these general schedulability analysis problems. However, each algorithm is problem specific and thus when a problem is modified, one has to develop new heuristics.

*This research was supported in part by ARO DAAG55-98-1-0393, ARO DAAG55-98-1-0466, AFOSR F49620-96-1-0204, NSF CCR-9619910, and ONR N00014-97-1-0505.

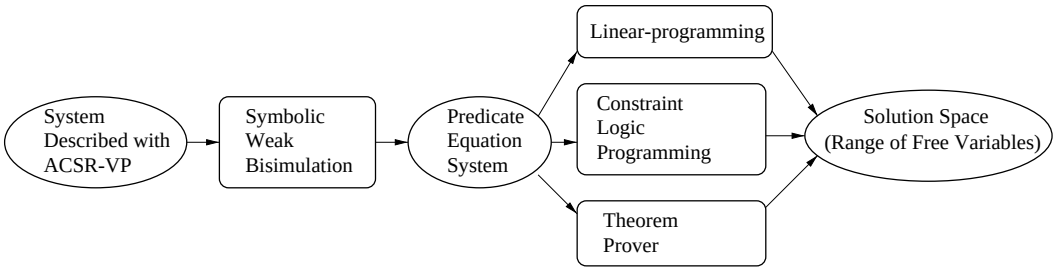


Figure 1: Overview of the Framework

In this paper, we describe a framework that allows one to model scheduling analysis problems with variable release and execution times, relative timing constraints, precedence relations, dynamic priorities, multiprocessors etc. Our approach is based on ACSR-VP and symbolic bisimulation algorithm.

ACSR (Algebra of Communicating Shared Resources) [14], is a discrete real-time process algebra. ACSR has several notions, such as resources, static priorities, exceptions, and interrupts, which are essential in modeling real-time systems. ACSR-VP is an extension of ACSR with value-passing and parameterized processes to be able to model real-time systems with variable timing attributes and dynamic priorities. In addition, symbolic bisimulation for ACSR-VP has been defined. ACSR-VP without symbolic bisimulation has been applied to the simple schedulability analysis problem [5], by assuming that all parameters are ground, i.e., constants. However, it is not possible to use the technique described in [5] to solve the general schedulability analysis problem with unknown timing parameters.

Figure 1 shows the overall structure of our approach. We specify a real-time system with unknown timing or priority parameters in ACSR-VP. For the schedulability analysis of the specified system, we check symbolically whether or not it is bisimilar to a process idling forever. The result is a set of predicate equations, which can be solved using widely available linear-programming or constraint-programming techniques. The solution to the set of equations identifies, if exists, under what values of unknown parameters the system becomes schedulable. To support the effective use of the the symbolic ACSR-VP analysis, we are developing a tool and planning to integrate into PARAGON [27], a toolset with graphical interface to support the use of ACSR.

The rest of the paper is organized as follows. Sections 2 overviews the theory of the underlying formal method, ACSR-VP, and introduce symbolic bisimulation for ACSR-VP expressions. Section 3 gives a specification of a scheduling problem, namely the *period assignment problem* and illustrates how to analyze an instances of this problem. Section 4 briefly describes the PARAGON toolset and its support for value-passing specifications, and outlines the incorporation of ACSR-VP into the toolset. We conclude with a summary and an outline of future work in Section 5.

2 ACSR-VP

ACSR-VP extends the process algebra ACSR [14] by allowing values to be communicated along communication channels. In this section we present ACSR-VP concentrating on its value-passing capabilities. We refer to the above papers for additional information on ACSR.

We assume a set of variables X ranged over by x, y, a set of values V ranged over by v, and a set of labels L ranged over by c, d. Moreover, we assume a set $Expr$ of expressions (which includes arithmetic expressions) and we let $BExpr \subset Expr$ be the subset containing boolean expressions. We let e and b range over $Expr$ and $BExpr$ respectively, and we write $\vec{z}$ for a tuple $z_1, \ldots z_n$ of syntactic entities.

ACSR-VP has two types of actions: instantaneous communication and timed resource access. Access to resources and communication channels is governed by priorities. A priority expression p is attached to every communication event and resource access. A partial order on the set of events and actions, the preemption relation, allows one to model preemption of lower-priority activities by higher-priority ones.

Instantaneous actions, called *events*, provide the basic synchronization and communication primitives in the process algebra. An event is denoted as a pair (i, e_p) representing execution of action i at priority e_p, where i ranges over τ, the idle action, $c?x$, the input action, and $c!e$, the output action. We use $\mathcal{D}_E$ to denote the domain of events and let λ range over events. We use $l(\lambda)$ and $\pi(\lambda)$ to represent the label and priority, respectively, of the event λ; e.g., $l((c!x, p)) = c!$ and $l((c?x, p)) = c?$. To model resource access, we assume that a system contains a finite set of serially-reusable resources drawn from some set R. An action that consumes one tick of time is drawn from the domain $\boldsymbol{P}(R \times Expr)$ with the restriction that each resource is represented at most once. For example the singleton action $\{(r, e_p)\}$ denotes the use of some resource $r \in R$ at priority

level e_p. The action $\emptyset$ represents idling for one unit of time, since no resource is consumed. We let $\mathcal{D}_R$ to denote the domain of timed actions with A, B, to range over $\mathcal{D}_R$. We define $\rho(A)$ to be the set of the resources used by action A, for example $\rho(\{(r_1, p_1), (r_2, p_2)\}) = \{r_1, r_2\}$. We also use $\pi_r(A)$ to denote the priority level of the use of the resource r in the action A; e.g., $\pi_{r_1}(\{(r_1, p_1), (r_2, p_2)\}) = p_1$, and write $\pi_r(A) = 0$ if $r \notin \rho(A)$. The entire domain of actions is denoted by $\mathcal{D} = \mathcal{D}_R \cup \mathcal{D}_E$, and we let α, β range over $\mathcal{D}$. We let P, Q range over ACSR-VP processes and we assume a set of process constants ranged over by C. The following grammar describes the syntax of ACSR-VP processes:

$$\begin{array}{rcl} P & ::= & \mathrm{NIL} \mid A : P \mid \lambda.P \mid P + P \mid P \| P \mid \\ & & b \rightarrow P \mid P \backslash F \mid [P]_I \mid C(\vec{x}). \end{array}$$

In the input-prefixed process $(c?x, e).P$ the occurrences of variable x is bound. We write $\mathbf{fv}(P)$ for the set of free variables of P. Each agent constant C has an associated definition $C(\vec{x}) \stackrel{\text{def}}{=} P$ where $\mathbf{fv}(P) \subseteq \vec{x}$ and $\vec{x}$ are pairwise distinct. We note that in an input prefix $(c?x, e).P$, e should not contain the bound variable x, although x may occur in P.

An informal explanation of ACSR-VP constructs follows: The process NIL represents the inactive process. There are two prefix operators, corresponding to the two types of actions. The first, $A : P$, executes a resource-consuming action during the first time unit and proceeds to process P. On the other hand $\lambda.\, P$, executes the instantaneous event λ and proceeds to P. The process $P + Q$ represents a nondeterministic choice between the two summands. The process $P \| Q$ describes the concurrent composition of P and Q: the component processes may proceed independently or interact with one another while executing instantaneous events, and they synchronize on timed actions. Process $b \rightarrow P$ represents the conditional process: it performs as P if boolean expression b evaluates to *true* and as NIL otherwise. In $P \backslash F$, where $F \subseteq L$, the scope of labels in F is restricted to process P: components of P may use these labels to interact with one another but not with P's environment. The construct $[P]_I$, $I \subseteq R$, produces a process that reserves the use of resources in I for itself, extending every action A in P with resources in $I - \rho(A)$ at priority 0.

The semantics of ACSR-VP processes may be provided as a labeled transition system, similarly to that of ACSR. It additionally makes use of the following ideas: Process $(c!e_1, e_2).P$ transmits the value obtained by evaluating expression e_1 along channel c, with priority the value of expression e_2, and then behaves like P. Process $(c?x, p).P$ receives a value v from communication channel c and then behaves like $P[v/x]$, that is P with v substituted for variable x. In the concurrent composition $(c?x, p_1).P_1 \| (c!v, p_2).P_2$, the two components of the parallel composition may synchronize with each other on channel c resulting in the transmission of value v and producing an event $(\tau, p_1 + p_2)$.

2.1 Unprioritized Symbolic Graphs with Assignment

Consider the simple ACSR-VP process $P \stackrel{\text{def}}{=} (in?x, 1).(out!x, 1).\mathrm{NIL}$ that receives a value along channel in and then outputs it on channel out, and where x ranges over integers. According to traditional methods for providing semantic models for concurrent processes, using transition graphs, process P in infinite branching, as it can engage in the transition $(in?n, 1)$ for every integer n. As a result standard techniques for analysis and verification cannot be applied to such processes.

Several approaches have been proposed to deal with this problem for various subclasses of value-passing processes [9, 16, 20, 13]. One of these advocates the use of *symbolic* semantics for providing finite representations of value-passing processes. This is achieved by taking a more conceptual view of value-passing than the one employed above. More specifically consider again process P. A description of its behavior can be sufficiently captured by exactly two actions: an input of an integer followed by the ouput of this integer. Based on this idea the notion of symbolic transition graphs [9] and transition graphs with assignment [16] were proposed and shown to capture a considerable class of processes.

In this section we present symbolic graphs with assignment for ACSR-VP processes. As it is not the intention of the paper to present in detail the process-calculus theory of this work, we only give an overview of the model and we refer to [13] for a complete discussion.

2.2 Symbolic Graph with Assignment

The notion of a $substitution$, which we also call $assignment$, is defined as follows. A *substitution* is any function $\theta \colon X \rightarrow Expr$, such that $\theta(x) \neq x$ for a finite number of $x \in X$. Given a substitution θ, the *support* (or *domain*) of θ is the set of variables $D(\theta) = \{x \mid \theta(x) \neq x\}$. A substitution whose support is empty is called the *identity substitution*, and is denoted by Id. When $|D(\theta)| = 1$, we use $[\theta(x)/x]$ for the substitution θ. Given two substitutions θ and σ, the *composition* of θ and σ is the substitution denoted by $\theta; \sigma$ such that for every variable x, $\theta; \sigma(x) = \sigma(\theta(x))$. We often write $\theta\sigma$ for $\theta; \sigma$.

An SGA is a rooted directed graph where each node n has an associated finite set of free variables $\mathbf{fv}(n)$ and each edge is labeled by a guarded action with assignment [16, 23]. Note that a node in SGA is a ACSR-VP term.

Definition 2.1 (SGA) A Symbolic Graph with Assignment (SGA) for ACSR-VP is a rooted directed graph where each node n has an associated ACSR-VP term and each edge is labeled by boolean, action, assignment, (b, α, θ). □

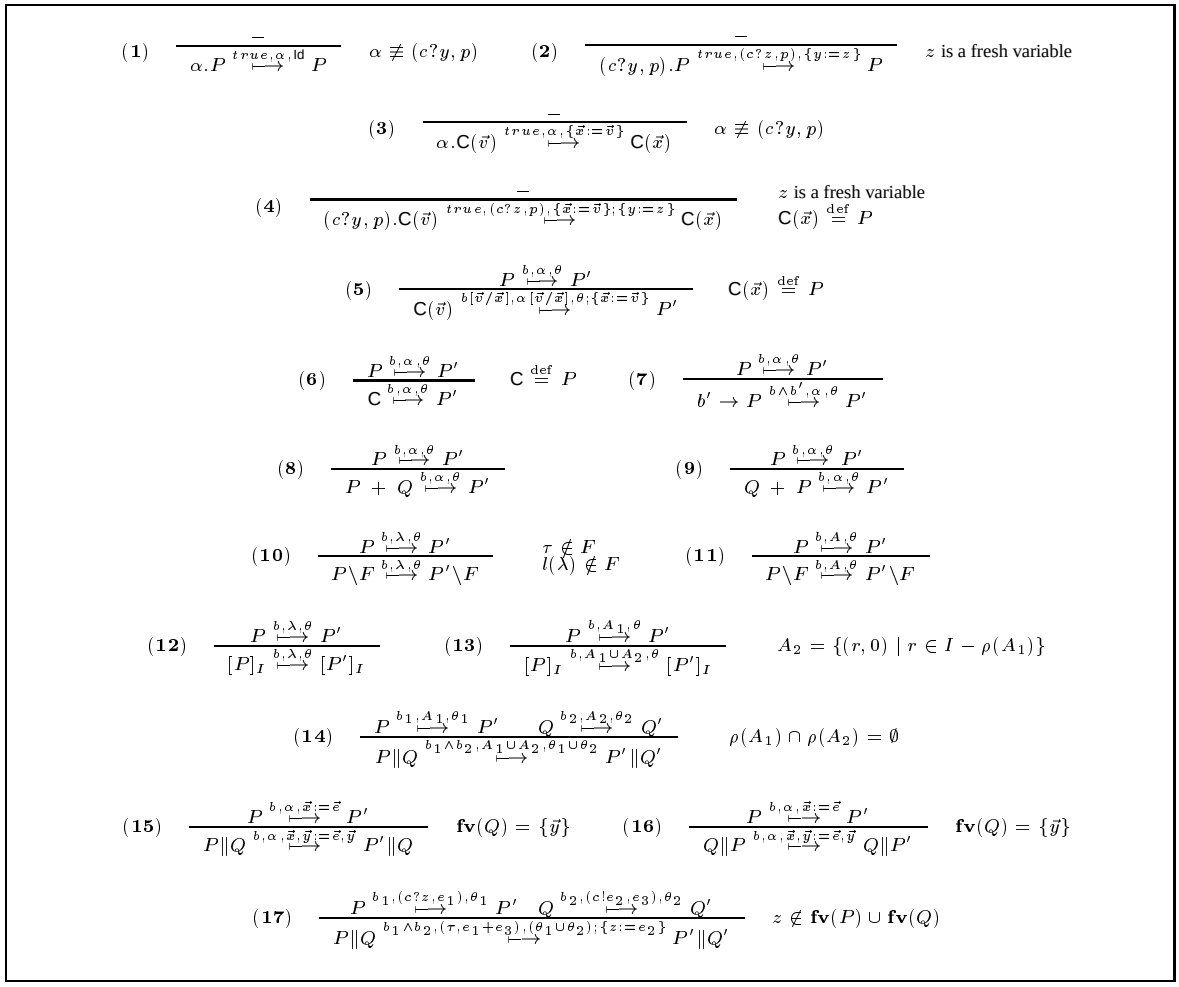
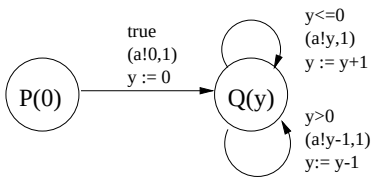


Figure 2: Rules for constructing Symbolic Graphs with Assignment

Given an ACSR-VP term, a SGA can be generated using the rules in Figure 2. Transition $P \overset{b,\alpha,\theta}{\longmapsto} P'$ denotes that given the truth of boolean expression b, P can evolve to P' by performing actions α and putting into effect the assignment θ. The interpretation of these rules is straightforward and we explain them by an example: Consider the following process. Process $P(0)$ can output the sequence of events $a!0$ infinitely many times.

$$\begin{array}{lcl} P(x) & \overset{\text{def}}{=} & (a!x,1).Q(x) \\ Q(y) & \overset{\text{def}}{=} & (y \le 0) \to (a!y,1).Q(y+1) \\ & + & (y > 0) \to (a!y-1,1).Q(y-1) \end{array}$$

Following SGA represents the process $P(0)$.



One possible interpretation of our SGA can be given along the lines of programming languages: Process P can be thought of as a procedure, so that $P(0)$ represents a call to P with actual parameter 0 which is accepted by P with formal parameter x declared in P's body. According to its definition, P outputs $a!0$ and calls process Q with actual parameter 0. Process Q then checks the validity of condition $y \leq 0$ or $y > 0$. If $y \leq 0$ is satisfied, process Q outputs $a!0$ and calls Q with actual parameter $y + 1$, where the value of y is 0 in this case. Similar reasoning can be applied for the condition $y > 0$. We believe that this interpretation, being similar to that of function calls and parameter passing in programming languages, is an intuitive way of interpreting the ACSR-VP terms.

2.3 The prioritized Symbolic Transition System

We have illustrated how ACSR-VP processes can be given finite representations as SGA's via the symbolic transition relation $\mapsto$. However, this relation makes no arbitration between actions with respect to their priorities. To achieve this, we refine the relation $\mapsto$ to obtain the prioritized symbolic transition system $\mapsto_\pi$. This is based on the notion of *preemption* which incorporates our treatment of priority, and in particular on relation $\succ$, the *preemptive relation*, a transitive, irreflexive relation on actions [2]. Then for two actions α and β, $\alpha \succ \beta$ denotes that α preempts β, which implies that in any real-time system, if there is a choice between the two actions, α will always be executed. For example $(c?x, 2) \succ (c?x, 1)$ and $\{(r, 2)\} \succ \{(r, 0)\}$.

Extending the notion of preemption in the value-passing setting involves dealing with the presence of free variables in process descriptions. For example, given actions $\alpha = (c?x, y_1)$ and $\beta = (c?x, y_2)$, whether $\alpha \succ \beta$ or $\beta \succ \alpha$ depends on the values to which variables y_1 and y_2 are instantiated. This idea can easily be incorporated to yield the prioritized transition relation $\mapsto_\pi$. For the precise definition we refer the reader to [13]. We illustrate this with an example. Consider process P:

$$\begin{array}{lcl} P(x) & \stackrel{\text{def}}{=} & (a?y, 1).P'(x, y) \\ P'(x, y) & \stackrel{\text{def}}{=} & (y \leq 1) \rightarrow (a!(x + y), y).NIL \\ & + & (y \leq 2) \rightarrow (a!(x + y), 2).NIL \end{array}$$

Figure 3 shows the unprioritized SGA for P and its prioritized version, Q. Note that transition $P' \stackrel{y \leq 1, (a!(x+y), y), \mathsf{Id}}{\longmapsto} NIL$ is preempted by $P' \stackrel{y \leq 2, (a!(x+y), 2), \mathsf{Id}}{\longmapsto} NIL$ since whenever the former is enabled, the latter is also enabled with a higher priority (that is, whenever $y \leq 1$, we have $y \leq 2$ and $y < 2$).

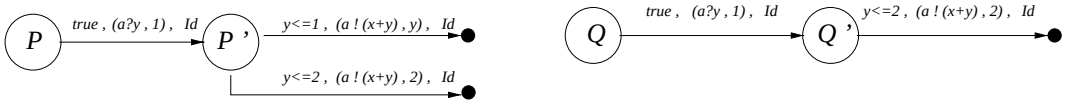


Figure 3: SGA of P and Q

2.4 Weak Bisimulation

Various methods have been proposed for the verification of concurrent processes. Central among them is observational equivalence that allows to compare an implementation with a specification of a given system. Observational equivalence is based on the idea that two equivalent systems exhibit the same behavior at their interfaces with the environment. This requirement was captured formally through the notion of *bisimulation* [19], a binary relation on the states of systems. Two states are bisimilar, if for each single computational step of the one, then there exists an appropriate matching (multiple) step of the other, leading to bisimilar states.

In this setting, bisimulation for symbolic transition graphs is defined in terms of relations parameterized on boolean expressions, of the form $\simeq^b$, where $p \simeq^b q$ if and only if, for each interpretation satisfying boolean b, p and q are bisimilar in the traditional notion. In [13] the authors have proposed weak version of bisimulations for SGA's, that is observational equivalences that abstract away from internal system behavior (both for late and early semantics). Furthermore, algorithms were presented for computing these equivalences. Given two closed processes whose symbolic transition graphs are finite, the algorithm constructs a predicate equation system that corresponds to the most general condition for the two processes to be weakly bisimilar.

Recall process $P(x)$ from Section 2.3. Furthermore, consider the following process with bound variable x':

$$
\begin{array}{lcl}
R(x') & \stackrel{\text{def}}{=} & (a?y', 1).R'(x', y') \\
R'(x', y') & \stackrel{\text{def}}{=} & (y' \leq 2) \rightarrow (a!(x' + y' + 1), 2).NIL
\end{array}
$$

The prioritized SGA for R is similar to Q with the exception that after receiving a value via channel a, R outputs value $x' + y' + 1$. Applying the symbolic bisimulation algorithm for processes P and R, we obtain the following predicate equation system.

$$
\begin{array}{lcl}
X_{00}(x, x') & = & \forall z X_{11}(z, x, x') \\
X_{11}(z, x, x') & = & z \leq 2 \rightarrow z \leq 2 \wedge x + z = x' + z + 1 \\
& \wedge & z \leq 2 \rightarrow z \leq 2 \wedge x' + z + 1 = x + z
\end{array}
$$

This equation system can easily be reduced to the equation $X_{00}(x, x') \equiv x = x' + 1$, which allows us to conclude that $P(x)$ and $R(x')$ are bisimilar if and only if $x = x' + 1$ holds. In general, since we are dealing with a domain of linear expressions, predicate equations obtained from the bisimulation algorithm can be solved using integer programming techniques [24].

3 Real-time Scheduling Problems

In this section, we show how a problem of real-time system scheduling can be specified and analyzed using ACSR-VP. According to [29], real-time scheduling problems can be categorized into the following three groups: priority assignment, execution synchronization, and schedulability analysis problems. The priority assignment problem requires assigning priorities to jobs so that the system schedulability is maximized. The execution synchronization problem is the problem of deciding when and how to release jobs so that the precedence constraints are satisfied and the system schedulability, as well as other performance concerns, are optimized. Schedulability analysis problem is the problem of verifying that a system is schedulable, given a certain priority assignment method and execution synchronization method.

Classic examples of solutions to these problems include the rate-monotonic priority assignment problem on a single processor [17]. It uses static priority assignment, where the priority of each job is assigned in the inverse order of period; a job with the shortest period has the highest priority. Deadline-monotonic priority assignment was proposed by [15], where the system has jobs with arbitrary relative deadlines.

The same groups of problems can be considered in the presence of end-to-end scheduling constraints. Gerber *et al.* [7] proposed the method to guarantee a system's end-to-end requirements of real-time systems. In [30], Tindell *et al.* attempted to compute upper bounds on the end-to-end response time. They also proposed priority assignment in distributed system where jobs have end-to-end deadlines. In [1], Bettati studied the problem of scheduling a set of jobs with arbitrary release times and end-to-end deadlines.

Our Approach. We propose to address real-time scheduling problems by means of analysis based on ACSR-VP. In this approach, a specific instance of a problem is specified as an ACSR-VP expression and symbolically analyzed. Figure 4 shows the overall structure of our approach. Rectangles with thick lines represent tools, and ovals in them represents the functional blocks inside tools. Rectangles with curved corner are text artifacts used as input/output for tools. We specify scheduling problems in the real-time system with unknown timing or priority parameters in the restricted form of ACSR-VP. The restricted form of ACSR-VP is defined to ensure that resulting SG (Symbolic Graph without assignment) derived from SGA is finite.

With a given set of ACSR-VP processes in the restricted form, the SGA is generated to capture the semantics of model. There are two paths that lead us to a solution. With the first path, we can generate the finite SG from the SGA and check bisimilarity with an infinite idle process. The result is a boolean expression with unknown parameters. This kind of boolean expression can be solved using integer programming, e.g., Omega Test [21], to find all solutions of the parameters. With the second path, the generated SGA is checked symbolically whether it is bisimilar to an idling process. Here, the result is a set of predicate equations with unknown parameters. This resulting set of equations can then be translated into a constraint logic program or into a boolean formula.

For a real-time scheduling problem, if a solution to a boolean expression or to the set of predicate equations exists, then it identifies under what values of unknown parameters the system becomes schedulable. Thus, the schedulability analysis is performed symbolically. For instance, in the rate-monotonic scheduling shown below, we want to find the periods of jobs to guarantee that a system can be scheduled. We call this problem the *period assignment* problem. In this problem, we let periods be free variables and describe a system as ACSR-VP process terms. These free variables appear in the resulting boolean expression or predicate equations that are generated from the bisimulation algorithm. Solutions for free variables represent the valid ranges of periods of the jobs, which make the system schedulable.

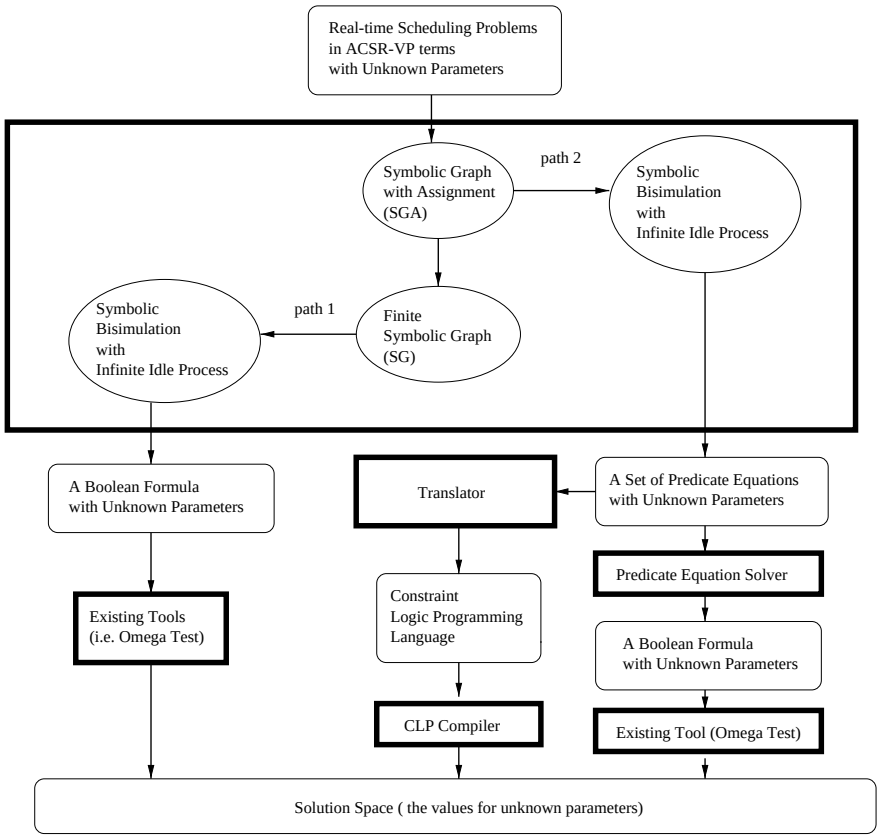


Figure 4: Our Approach to Real-time Scheduling Problem

Our method is expressive to model complex real-time systems in general. Furthermore, it is effective in the sense that the resulting boolean formulas and predicate equations can be solved efficiently. For instance, there has been active research [6] to solve boolean formulas efficiently, and there are existing tools such as omega test [21], which are very fast in practice. For predicate equations, there are constraint programming techniques that are known for solving linear (in)equation constraints efficiently [10, 24]. Furthermore, the size of the SGAs constructed from ACSR-VP terms is significantly smaller than that of Labeled Transition Systems(LTS) constructed from ACSR. Consequently, this greatly reduces the state explosion problem, and thus, we can now model larger systems and solve problems which are not possible using ACSR (and its toolset called PARAGON) due to state explosion.

We now illustrate our approach by showing how to solve a rate-monotonic scheduling problem, known as the *period assignment problem*. Our method of solving this problem is optimal in the sense that if the method can not find a period assignment, then the system cannot be scheduled for any assignment of periods.

Period Assignment Problem for Rate Monotonic Scheduling. We briefly state how rate monotonic scheduling works and show our approach to the period assignment problem. Rate monotonic scheduling is a preemptive static priority driven scheduling algorithm, which works as follows. The priorities of tasks are assigned in the reverse order of lengths of their periods, that is, tasks with shorter periods are assigned higher priorities than tasks with longer periods. Scheduling decisions are made whenever any task becomes ready or whenever a processor becomes idle. At each scheduling decision time, a ready task with the highest priority is executed. The following ACSR-VP process describes a job with unknown period:

$$
\begin{array}{lcll}
Job_i(p_i,s_i,t_i) & \stackrel{\text{def}}{=} & (s_i < E_i) \wedge (t_i < D_i) & \rightarrow \{(cpu, MAX - p_i)\} : Job_i(p_i, s_i+1, t_i+1) \\
 & & & + \quad \emptyset : Job_i(p_i, s_i, t_i+1) \\
 & + & (s_i = E_i) \wedge (t_i \leq D_i) & \rightarrow Wait(p_i, t_i) \\
Wait_i(p_i,t_i) & \stackrel{\text{def}}{=} & (t_i \leq P_{i,max}) \wedge (t_i < p_i) & \rightarrow \emptyset : Wait_i(p_i, t_i+1) \\
 & + & (t_i \leq P_{i,max}) \wedge (t_i = p_i) & \rightarrow (\tau, 1).Job_i(p_i, 0, 0)
\end{array}
$$

where E_i and D_i represent the constant values for the execution time and deadline of Job_i, respectively. Process $Job_i\ (p_i, s_i, t_i)$ represents a job with period p_i, which has accumulated s_i units of processing time in the current period. The current period has started t_i time units ago. As long as the job is not finished ($s_i < E_i$) and the current deadline is not over ($t_i < D_i$), the job competes with other jobs for access to the *cpu* resource. The priority of Job_i is $MAX - p_i$, where MAX is the largest possible period. That is, the job with shortest period has the highest priority. If the job is preempted by a higher-priority process, it idles in that time unit. Alternatively, if the job has completed ($s_i = E_i$), it turns into the $Wait_i(p_i, t_i)$ process, which idles until the end of the current period and restarts itself. $P_{i,max}$ represents the possible maximum value for the period of Job_i. In this rate monotonic setting, priorities are unknown since the period of each job is not known.

For a job Job_i where period is known to be P_i, it can be described as follows:

$$\begin{array}{lcll} Job_i(s_i,t_i) & \stackrel{\text{def}}{=} & (s_i < E_i) \wedge (t_i < D_i) & \rightarrow \quad \{(cpu, MAX - P_i)\} : Job_i(s_i+1, t_i+1) \\ & & & + \quad \emptyset : Job_i(s_i, t_i+1) \\ & + & (s_i = E_i) \wedge (t_i \leq D_i) & \rightarrow \quad Wait(t_i) \\ Wait_i(t_i) & \stackrel{\text{def}}{=} & (t_i < P_i) & \rightarrow \quad \emptyset : Wait_i(t_i+1) \\ & + & (t_i = P_i) & \rightarrow \quad (\tau, 1).Job_i(0,0) \end{array}$$

Assuming that, initially, all jobs start at time 0, we can capture the behavior of the whole system as follows.

$$RM(\vec{p}) \quad \stackrel{\text{def}}{=} \quad [Job_1 \parallel \cdots \parallel Job_n]_{\{cpu\}}$$

where Job_i, $i \in \{1, \ldots, n\}$ can be either $Job_i(p_i, 0, 0)$ if the period is unknown or $Job_i(0,0)$ otherwise. Symbolic weak bisimulation relation with infinite idle process can be checked by applying the algorithm shown in [13]. The result is a set of predicate equations or a boolean expression if we translate the SGA into the SG. A boolean expression can be solved automatically by the existing integer programming tool.

4 PARAGON Toolset

Our approach to the symbolic analysis of ACSR-VP specifications can be applied effectively to non-trivial problems only if there are good tool supports for specifications in ACSR-VP and analysis algorithms described in the preceding sections. Their usefulness will be enhanced if this tool support is provided within an extensive specification and verification framework, where symbolic analysis can be supplemented by other analysis techniques. Such framework can be provided by extending PARAGON [27], a toolset based on ACSR and other related formalisms.

PARAGON is a toolset for the specification and analysis of distributed resource-bound real-time systems. PARAGON supports both graphical and textual input. Graphical specifications enhance the usability of a formal model, giving a visual representation of hierarchy modules in the system and of interconnections between modules. Graphical specifications in PARAGON are expressed using the GCSR language, based on a real-time process algebra. A GCSR specification is a collection processes, which consist of nodes, connected by edges. The execution of the system proceeds from node to node along the edges. There are several types of nodes to express sequential behavior of a system module and its resource requirements. In addition to these, a *compound* node provides hierarchy. One or more parallel processes can be placed into a compound node. Interactions between processes in a compound node can be made local to the node, that is, invisible to the processes outside.

For analysis, PARAGON supports several techniques: extensive checking of syntactic consistency constraints; state space exploration, including reachability analysis and deadlock detection; checking equivalence between two specifications; and visual simulation.

PARAGON already supports parameterization in specifications and can deal with value passing. This enables concise specification of arrays of similar components, multiple resources of the same type, and value passing between processes. Event and resource names and process references in an indexed specification can contain multiple indices. Indices may be represented as integers or integer-valued expressions using index variables. The syntax of the current parameterized specifications, although slightly different from that of ACSR-VP, provides for an easy translation between the two formalisms. However, parametric treatment of data values is currently missing in PARAGON. Every parameterized PARAGON specifications is equivalent to a non-parameterized one, and handling of parameterization during analysis is done through an "un-parameterizing" translation. This approach is very inefficient, as it creates a separate process for every instantiation of free index variables in the parameterized process, many of which are not necessary for the subsequent analysis. Therefore, it is necessary to use a better internal representation that handles index variables symbolically, such as SGA described in this paper.

5 Conclusions

We have described a formal framework for the specification and analysis of real-time scheduling problems. Our framework is based on ACSR-VP and symbolic bisimulation. The major advantage of our approach is that the same framework can be used for scheduling problems with different assumptions and parameters. In other scheduling-theory based approaches, new analysis algorithms need to be devised for problems with different assumptions since applicability of a particular algorithm is limited to specific system characteristics.

We believe that ACSR-VP is expressive enough to model any real-time system. In particular, our method is appropriate to model many complex real-time systems and can be used to solve the *priority assignment problem*, *execution synchronization problem*, *end-to-end design problem*, and *schedulability analysis problem*. It depends on light-weight formal methods in the sense that resulting predicate equation systems can be solved with existing techniques such as linear programming or constraint programming, which can be solved using linear equation constraints efficiently in practice [24].

The novel aspect of our approach is that parametrized design of a real-time system can be described formally and analyzed automatically, all within a process-algebraic framework. It has often been noted that scheduling work is not adequately integrated with other aspects of real-time system development [3]. Our work is a step toward such an integration, which helps to meet our goal of making the timed process algebra ACSR a useful formalism for supporting the development of reliable real-time systems. Our approach allows the same specification to be subjected to the analysis of both schedulability and functional correctness.

There are several issues that we need to address to make our approach practical. We showed that resulted predicate equation systems can be solved with constraint logic programming or linear programming, but they can be rather complicated. We plan to investigate when resulting equation systems become easy or difficult to solve. In the worst case, we may have to use a more powerful technique such as theorem prover; however, it is not clear whether any reasonable real-time system scheduling problem can result in such a complex equation system. We are currently augmenting PARAGON, the toolset for ACSR, to support the full syntax of ACSR-VP directly and implementing a symbolic bisimulation algorithm. This toolset will allow us to experimentally evaluate the effectiveness of our approach with a number of large scale real-time systems.

References

[1] R. Bettati. *End-to-end Scheduling to Meet Deadlines in Distributed Systems.* PhD thesis, University of Illinois at Urbana-Champaign, 1994.

[2] P. Brémond-Grégoire, I. Lee, and R. Gerber. ACSR: An Algebra of Communicating Shared Resources with Dense Time and Priorities. In *Proc. of CONCUR '93*, 1993.

[3] A. Burns. Preemptive priority-based scheduling: An appropriate engineering approach. In Sang H. Song, editor, *Advances in Real-Time Systems*, chapter 10, pages 225–248. Prentice Hall, 1995.

[4] M. Chen and K. Lin. Dynamic Priority Ceilings: A Concurrency Control Protocol for Real-Time Systems. *Real-Time Systems*, 2(4):325–346, 1990.

[5] J-Y. Choi, I. Lee, and H-L Xie. The Specification and Schedulability Analysis of Real-Time Systems using ACSR. In *Proc. of IEEE Real-Time Systems Symposium*, December 1995.

[6] Uffe Engberg and Kim S. Larsen. Efficient Simplification of Bisimulation Formulas. In *Proceedings of the Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, pages 111–132. LNCS 1019, Springer-Verlag, 1995.

[7] Richard Gerber, Dongin Kang, Seongsoo Hong, and Manas Saksena. End-to-End Design of Real-Time Systems. In D. Mandrioli and C. Heitmeyer, editors, *Formal Methods in Real-Time Computing*. John Wiley & Sons, 1996.

[8] Constance Heitmeyer and Dino Mandrioli. *Formal Methods for Real-Time Computing*. Jonh Wiley and Sons, 1996.

[9] M. Hennessy and H. Lin. Symbolic bisimulations. *Theoretical Computer Science*, 138:353–389, 1995.

[10] Joxan Jaffar and Michael J. Maher. Constraint logic programming: A survey. *Journal of Logic Programming*, 19(20):503–581, 1994.

[11] M. Joseph and P. Pandya. Finding Response Times in a Real-Time System. *Computer Journal*, 29(5):390–395, 1986.

[12] Mathai Joseph. *Real-Time Systems: Specification, Verification and Analysis*. Prentice Hall Intl., 1996.

[13] Hee-Hwan Kwak, Jin-Young Choi, Insup Lee, and Anna Philippou. Symbolic weak bisimulation for value-passing calculi. Technical Report MS-CIS-98-22, University of Pennsylvania, Department of Computer and Information Science, 1998.

[14] I. Lee, P. Brémond-Grégoire, and R. Gerber. A Process Algebraic Approach to the Specification and Analysis of Resource-Bound Real-Time Systems. *Proceedings of the IEEE*, pages 158–171, Jan 1994.

[15] J. Leung and J. Whitehead. On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation*, pages 2:237–250, 1982.

[16] H. Lin. Symbolic graphs with assignment. In U.Montanari and V.Sassone, editors, *Proceedings CONCUR 96*, volume 1119 of *Lecture Notes in Computer Science*, pages 50–65. Springer-Verlag, 1996.

[17] C. L. Liu and J. W. Layland. Scheduling Algorithms for Multi-programming in A Hard-Real-Time Environment. *Journal of the Association for Computing Machinery*, 20(1):46 – 61, January 1973.

[18] J. W. S. Liu and R. Ha. Efficient methods of validating timing constraints. In Sang H. Song, editor, *Advances in Real-Time Systems*, chapter 9, pages 199–233. Prentice Hall, 1995.

[19] R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

[20] P. Paczkowski. Characterizing bisimilarity of value-passing parameterised processes. In *Proceedings of the Infinity Workshop on Verification of Infinite State Systems*, pages 47–55, 1996.

[21] William Pugh. The Omega test: a fast and practical integer programming algorithm for dependence analysis. *Communications of the ACM*, 8:102–114, August 1992.

[22] R. Rajikumar, L. Sha, and J. Lehoczky. Real-Time Synchronization Protocols for Multiprocessors. In *Proc. of IEEE Real-Time Systems Symposium*, pages 259–272, 1989.

[23] Julian Rathke. *Symbolic Techniques for Value-passing Calculi*. PhD thesis, University of Sussex, 1997.

[24] Romesh Saigal. *Linear Programming : A Modern Integrated Analysis*. Kluwer Academic Publishers, 1995.

[25] L. Sha, R. Rajkumar, and J. Lehoczky. Priority Inheritance Protocols: An Approach to Real-time Synchronization. *IEEE Transactions on Computers*, 39(9):1175–1185, September 1990.

[26] L. Sha, R. Rajkumar, J. Lehoczky, and K. Ramamritham. Mode change Protocols for Priority Driven Preemptive Scheduling. *Real-Time Systems: The International Journal of Time Critical Computing Systems*, 1(3), December 1989.

[27] O. Sokolsky, I. Lee, and H. Ben-Abdallah. Specification and analysis of real-time systems with paragon. *Annals of Software Engineering*, 1999. To appear.

[28] B. Sprunt, L. Sha, and J. Lehoczky. Aperiodic Task Scheduling for Hard-Real-Time Systems. *Real-Time Systems: The International Journal of Time Critical Computing Systems*, 1(1):27–60, 1989.

[29] Jun Sun. *Fixed-priority End-to-end Scheduling in Distributed Real-time Systems*. PhD thesis, University of Illinois at Urbana-Champaign, 1997.

[30] K. Tindell and J. Clark. Holistic Schedulability Analysis for Distributed Hard Real-time Systems. *Microprogramming*, 50(2):117–134, April 1994.